ROBUY 3203 - Final Project

– Monocular Depth Estimation –

submitted by
Niraj Pudasaini, Aris Panousopoulos

August 8, 2024

Professor
Felix Juefei Xu

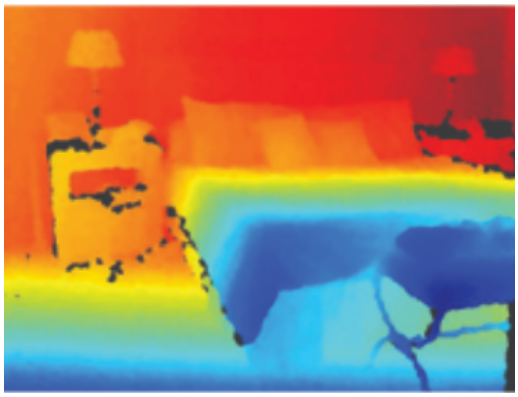
Instructor
Irving Fang

Contents

1 Introduction

Monocular depth estimation is the task of predicting the depth value of each pixel or inferring general depth information from a single RGB image. This is an important task in computer vision, particularly in the field of autonomous robots. However, obtaining accurate depth information from a single image remains a challenge due to the lack of geometric solutions[1].

(a) Real sample image



(b) Pixel-wise depth information

The goal of this project is to perform monocular depth estimation to predict depth value of each pixel or infer general depth information from a single RGB image. The motivation behind this project is to develop a custom approach to monocular depth estimation that achieves high accuracy on the test partition of the NYU-Depth V2 dataset while also demonstrating generalizability on a hidden dataset. In the example of the left picture, it shows RGB images, while the second image displays depth information maps corresponding to the images. By estimating depth from a single image, it is possible to reconstruct a 3D model of the scene, which has many applications in fields such as autonomous robots, augmented reality, and virtual reality.

2 Background

2.1 Monocular vs. Stereo Depth Estimation

Depth estimation techniques can generally be categorized into two main types: monocular and stereo.Monocular depth estimation relies on a single RGB image to infer depth information, while stereo depth estimation leverages two or more images captured from different viewpoints to estimate depth.Stereo depth estimation is typically more accurate and easier to implement, as it exploits the disparity between corresponding points in the two images to compute depth. However, stereo systems require multiple cameras and pre-

cise calibration, which can be expensive and challenging to set up and maintain, especially in dynamic environments.

On the other hand, monocular depth estimation, as used in our project, relies on a single image, making it a more affordable and compact solution. However, this also makes the task more challenging, as it requires learning complex features and patterns in the image to infer depth information. Recent advances in deep learning have led to significant improvements in monocular depth estimation, making it an increasingly attractive option for robotic applications. This project focuses on monocular depth estimation due to its potential for broader applicability and reduced hardware requirements compared to stereo depth estimation.

3 Datasets - Loading and Preprocess

The NYU-Depth V2 dataset is a large collection of RGB and depth images captured in indoor scenes using the Microsoft Kinect sensor. It contains 18,198 labeled pairs of aligned RGB and depth images for training and evaluating depth estimation models.

In this section, we describe the process of loading and preprocessing the dataset for our depth estimation task. We implement a custom dataset class for handling RGB and depth images and utilize PyTorch's DataLoader for efficient loading and batching.

3.1 Data List Generation

The function `get_datalist(data_dir)` generates a list of tuples containing the paths to the RGB and depth images for each scene in the dataset.

3.2 Custom Dataset Class

We implement a custom dataset class named `my_dataset` that inherits from PyTorch's `Dataset` class. This custom dataset class is responsible for loading and transforming the RGB and depth images. It has three main components:

- `data_list`: A list of tuples containing the paths to the RGB and depth images for each scene
- `rgb_transform`: A torchvision transformation pipeline for preprocessing the RGB images
- `depth_transform`: A torchvision transformation pipeline for preprocessing the depth images

The `my_dataset` class overrides the `__len__` and `__getitem__` methods from the base `Dataset` class. The `__len__` method returns the total number of data samples, while the `__getitem__` method loads and transforms the RGB and depth images for a given index.

3.3 Calculating Mean and Standard Deviation

The function `calculate_mean_std(datalists, sample_size)` calculates the mean and standard deviation of the pixel values of a 30% random sample of images from the dataset. This information is used for normalizing the images and depths before feeding into the network.

3.4 Data Augmentation

Data preprocessing is a crucial step in ensuring that the input data fed to the model is consistent and well-suited for training. It speeds up the trainin process as well. In this project, the following preprocessing steps were applied to the NYU-Depth V2 dataset:

1. Resizing: Both RGB and depth images were resized to a fixed resolution of 256x256 pixels, which is a suitable input size for the depth estimation model.
2. Normalization: The pixel values of the RGB images were normalized using the mean and standard deviation values calculated from a random sample of the dataset. The normalization helps the model generalize better by centering the input distribution and ensuring that different channels have similar value ranges.

4 Model

Our approach to monocular depth estimation is based on a U-Net-like architecture, which remains a popular choice for segmentation and can be used for depth estimation tasks. We developed a custom encoder and decoder, with feature concatenation between corresponding layers. The following sections provide a detailed explanation of the components and functions used in our implementation.

4.1 Encoder: Custom Encoder

The encoder part of our U-Net-like architecture consists of a series of 2D convolutional layers with kernel size 3 and padding 1, each followed by a ReLU activation function. The number of filters in these layers increases from 64 to 512, doubling with each subsequent layer. Max pooling with kernel size 2 and stride 2 is applied after the first layer and each subsequent layer to reduce the spatial dimensions.

4.2 Decoder: Custom Decoder with Transposed Convolutions

The decoder part of our architecture consists of a series of transposed convolutional layers, each followed by a ReLU activation function. These layers are responsible for upsampling the feature maps and increasing their spatial resolution. After each upsampling, the upsampled feature maps are concatenated with the corresponding feature maps from the encoder, and a 2D convolutional layer is applied. The number of filters in the transposed convolutional layers decreases from 512 to 64, halving with each subsequent layer.

4.3 Output Layer

The output layer of our depth estimation model is a 2D convolutional layer with a single output channel and kernel size 3 and padding 1. This layer is responsible for producing the final depth map from the features extracted and processed by the encoder and decoder.

5 Experiments and Results

5.1 Evaluation Metrics

Our model will be evaluated using the following metrics.

1. Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{|T|}\sum_{p\in T}(d_{gt}(p) - d_p(p))^2} \tag{1}$$

   where $d_{gt}(p)$ and $d_p(p)$ denote the ground truth and predicted depths at a given pixel p, respectively, and T denotes the set of pixels for which the ground truth values are available.

2. Absolute Relative Error (Abs Rel):

$$AbsRel = \frac{1}{|T|}\sum_{p\in T}\frac{|d_{gt}(p) - d_p(p)|}{d_{gt}(p)} \tag{2}$$

3. Threshold:

$$\text{Threshold} = \frac{1}{|T|}\sum_{p\in T}[\max(\frac{d_{gt}(p)}{d_p(p)}, \frac{d_p(p)}{d_{gt}(p)}) < \delta] \tag{3}$$

   where δ is a fixed threshold value, set to 1.25.

The Root Mean Squared Error (RMSE) measures the average squared difference between the predicted depth and ground truth depth values over all the pixels in the test set, while the Absolute Relative Error (Abs Rel) calculates the average absolute difference between the predicted depth and ground truth depth values, normalized by the ground truth depth. The Threshold metric computes the percentage of pixels whose predicted depth values fall within a certain threshold of the ground truth values, where the threshold is set to a fixed value of 1.25. Overall, these metrics collectively provide a good evaluation of the accuracy of depth prediction models and help to compare the performance of different models.

5.2 Training Results

During the training, the train loss and validation loss were trakced and finally plotted over each epochs. The following plot illustrates the results of our training process:

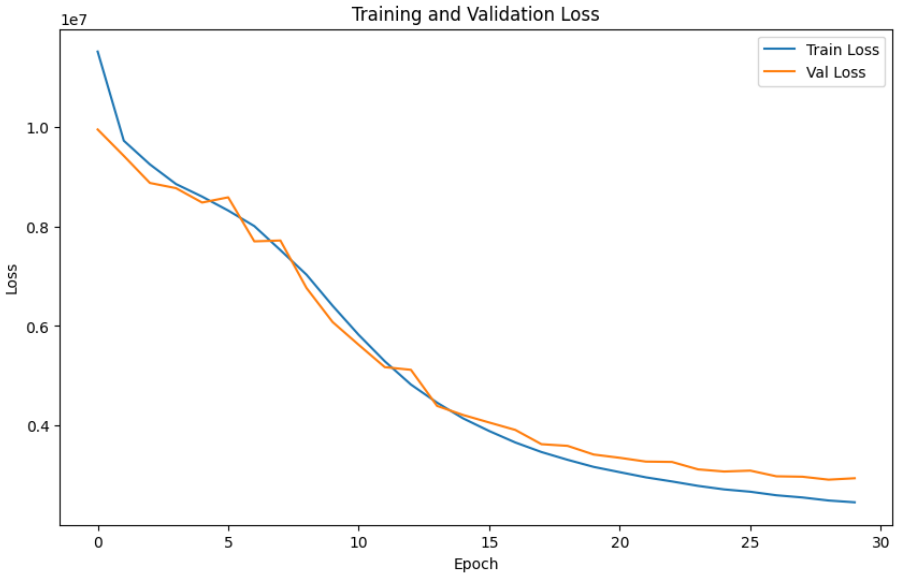


Figure 2: Plot of losses over epochs

We can see that the plot flatten outs at around the 30 epochs, which shows that our training till 30 epochs were good enough. It can also be seen that initially, the validation and training losses are somewhat similar, and after 15 epochs, the training loss is lower than validation loss, which implies that the model is starting to overfit the training data and not generalizing well to new data. Therefore, it might be beneficial to stop training the model around 30 epochs.

5.3 Evaluation Results

Based on the evaluation metrics defined on section 5, we performed the evaluation on the valid data set and also the training data set, which is shown in the figures below.

```
Root Mean Squared Error on Validation set: 1600.9570
Absolute Relative Difference on Validation set: -179.2963
Threshold check on Validation set: 81.51%
```

Figure 3: Evaluation result on Valid dataset

```
Root Mean Squared Error on Validation set: 1586.3662
Absolute Relative Difference on Validation set: -176.3393
Threshold check on Validation set: 81.55%
```

Figure 4: Evaluation result on Train dataset

As seen in figure 3 and 4, the threshold chcek on validation and test set set are both above 81%, which is decent compared to the results in the paper Monocular Depth Estimation Based On Deep Learning: An Overview [1].

However the values of the RMSE and Abs Rel are off. We suspected that this was because of the normalization of the depth data and lack of sigmoid like activation function at the output of the network. In other words, we suspected, the output of the neural network wasn't mapped to normalized depth data. We tried adding the sigmoid functions at the output layer, but our model didn't improve, but it hampered the model from learning, and the loss remained somewhat constant over epochs.

We also tried a different approach of using a pretrained Resnet model (labelled as "depth-resnet-model.py") in the submission. With different variations of models, and loss functions (including BerHu and l1 loss), we found that the UNET influenced architecture with root mean squared loss performed well, which is the model that we have included as the final model in our project.

5.4 Inference

The following results are the sample inference performed on the 5 samples of validation datasets. Both the normalized and de-normalized images are included. Visually, it looks like the model performed the job well enough. Overall, our model inferrered the data well, as also suggested from our threshold check results from validation dataset

Figure 5: Inference on the Dataset

References

[1] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, "Monocular depth estimation based on deep learning: An overview," pp. 1612–1627, 2020. [Online]. Available: https://doi.org/10.1007%2Fs11431-020-1582-8